

PHASE IV

PROJECT DESCRIPTION

GlobalRides, a company similar to Uber and Uber Eats, operates a platform connecting riders, drivers, food delivery customers, and restaurants. The company seeks to design a relational database to effectively manage its multifaceted operations, including Riders, Drivers, Customers, Restaurants, Rides, Food Orders, Payments, and Reviews.

A user can serve multiple roles, such as being a Rider, a Customer, a Driver, and a Restaurant Owner. All users share common attributes, including User ID, Name (First, Middle, Last), Contact Details, Address, Gender, and Date of Birth. A user may provide multiple contact numbers.

For Rides, each booking has attributes like Ride ID, Pickup and Drop-off Locations, Pickup Time, Ride Fare, and Payment Status. Rides are associated with Drivers, who can be assigned multiple rides, while each ride can only have one driver. Drivers have specific attributes such as Driver ID, License Details, Vehicle Information, and Experience.

For Food Orders, customers can place orders with Restaurants listed on the platform. Each order records Order ID, Restaurant ID, Customer ID, Order Date, Delivery Status, Total Amount, and Payment Method. Orders consist of multiple items from the restaurant's menu. Each restaurant manages its menu, including attributes like Item ID, Name, Description, Price, and Food Category (e.g., Appetizers, Main Course, Desserts). Restaurants have attributes such as Restaurant ID, Name, Address, Cuisine, Operational Hours, and Ownership details. Restaurants can also run promotions tied to specific menu items. These promotions are unique within the restaurant and include Promotion ID, Description, and Validity Period.

Riders and Customers can leave reviews. Reviews have attributes like Review ID, Rating, Feedback Text, and Date, linked to the specific ride, food item, or restaurant being reviewed.

Employees of GlobalRides are integral to the company's operations and are categorized into several distinct roles: Platform Managers, Support Agents, and Delivery Coordinators. Each employee has an Employee ID, a unique identifier following a predefined format such as "E####," where "####" represents a sequence of digits. Employees must be at least 18 years old. Additional general attributes include their Start Date, which records when the employee joined the organization, and Department, signifying the specific area within the company they are associated with. Beyond these common attributes, employees have role-specific details that define their responsibilities and qualifications.

Platform Managers are responsible for overseeing the overall system and managing high-level operations. Their role involves monitoring platform performance, addressing escalations, and ensuring seamless coordination across various aspects of the service. They play a pivotal role in strategic planning and decision-making to enhance the efficiency and functionality of the platform.

Support Agents are dedicated to handling user inquiries and ensuring customer satisfaction. Each inquiry is tracked with a unique ID and includes details such as the inquiry time and resolution status. To ensure effectiveness, Support Agents must undergo training conducted by certified Trainers, who can be either Platform Managers or Delivery Coordinators. Trainers possess unique Trainer Certificates, with the

issuance date of each certificate carefully recorded. One Trainer is capable of training multiple Support Agents, ensuring comprehensive coverage and skill development within the team.

Delivery Coordinators are tasked with managing the logistics of delivery drivers, focusing on ensuring timely and efficient order deliveries. They are responsible for assigning drivers to orders, optimizing routes, and addressing operational challenges such as delays or vehicle issues. Their role is crucial in maintaining the smooth flow of delivery operations and upholding service quality standards.

Each role is vital to the seamless operation of GlobalRides, and the database must capture these intricate relationships and dependencies. The employee hierarchy, training records, and role-based permissions need to be efficiently organized within the database to ensure clarity and facilitate operations across the company.

b. Project Questions

1. Would a superclass/subclass relationship be beneficial in the GlobalRides database design? Why or why not?

Yes, a superclass/subclass relationship is extremely beneficial in the GlobalRides database design for several key reasons:

The first major instance where this relationship helps us is with Users and user types. The requirements clearly show that users can take on multiple roles like being a Rider, Customer, Driver, and Restaurant Owner simultaneously. By setting up User as a superclass with common attributes (User ID, Name, Contact Details, etc.), we can create subclasses for each role that inherit these shared properties. These would be overlapping subclasses since one person can belong to multiple categories at once - exactly what we need for GlobalRides' flexible user roles.

The second clear application is for the Employee structure. Here, we have Platform Managers, Support Agents, and Delivery Coordinators who all share basic employee attributes but also have unique role-specific details. Creating an Employee superclass with these specific job types as disjoint subclasses makes perfect sense because an employee can only hold one of these positions at a time.

This approach gives us several advantages:

- We avoid duplicating data like names and contact info across multiple tables
- We can easily add new user or employee types later without redesigning everything
- Our database structure actually matches how the business works in real life
- We can run queries either across all users/employees or target specific types

For implementation, the User hierarchy would need a structure supporting overlapping roles, while the Employee hierarchy could use either a type indicator column or separate tables for each role with foreign keys back to the main Employee table.

This design choice helps us build a database that's both practical to use and accurately represents GlobalRides' business model.

2. Can you think of 5 additional rules (other than those described above) that would likely be used in this environment? How would your design change to accommodate these rules?

Here are five high-level business rules that would be valuable for the GlobalRides platform, along with simplified design changes needed to implement them:

1. Driver Verification and Safety Requirements

Rule: Drivers must pass background checks, maintain valid insurance, and undergo periodic vehicle inspections to remain active.

Key Design Changes:

- Add verification status fields to Driver table
- Create a DriverDocuments table to track licenses, insurance, and inspection records
- Implement status checks before ride assignments

2. Dynamic Pricing System

Rule: Ride fares fluctuate based on demand, time of day, weather conditions, and special events.

Key Design Changes:

- Add base price and multiplier fields to fare calculations
- Create tables to track surge pricing factors and conditions
- Implement pricing history for analytics and transparency

3. Customer Loyalty Program

Rule: Users earn points for rides and food orders that can be redeemed for discounts or free services.

Key Design Changes:

- Create tables for points tracking and transaction history
- Add point calculation fields to order and ride tables
- Implement tier-based status levels for regular customers

4. Geographic Service Zones

Rule: Service availability, restaurant offerings, and delivery fees vary by geographic zone with temporary restrictions possible.

Key Design Changes:

- Create zones table with boundaries and service types
- Link restaurants and drivers to specific zones
- Add zone-specific pricing and availability rules

5. Scheduled Driver Availability

Rule: Drivers can set preferred working hours, zones, and ride types to improve work-life balance.

Key Design Changes:

- Create driver schedule and preferences tables
- Implement availability status tracking
- Add location tracking for efficient ride assignment

These rules would enhance service quality, improve operational efficiency, and provide better experiences for all GlobalRides stakeholders.

3. Justify the use of a Relational DBMS like Oracle for this project.

For this project, we have used MySQL RDBMS.

Justification for MySQL RDBMS for GlobalRides

MySQL's relational database management system is ideal for GlobalRides for these critical reasons:

Data Integrity

GlobalRides requires strict accuracy in ride-driver assignments, order-restaurant connections, and payment processing. MySQL's robust constraints and ACID-compliant transactions ensure data consistency across these interconnected operations, preventing costly errors in financial transactions and service deliveries.

Complex Relationships

The platform features numerous complex relationships:

- Users with multiple overlapping roles
- Many-to-many connections between restaurants, menu items, and promotions
- Hierarchical employee structures

Relational databases excel at modeling these relationships through normalization and junction tables, with Oracle's efficient join operations retrieving related data effectively.

Query Capabilities

MySQL's advanced SQL implementation supports the complex queries needed for:

- Driver efficiency metrics
- Restaurant performance analysis
- Payment reconciliation
- Location-based service optimization

Security

With sensitive payment and personal data, Oracle provides:

- Role-based access control matching employee hierarchies
- Row-level security for restaurant-specific data
- Encryption for personal and payment information

Scalability

As GlobalRides grows, MySQL offers:

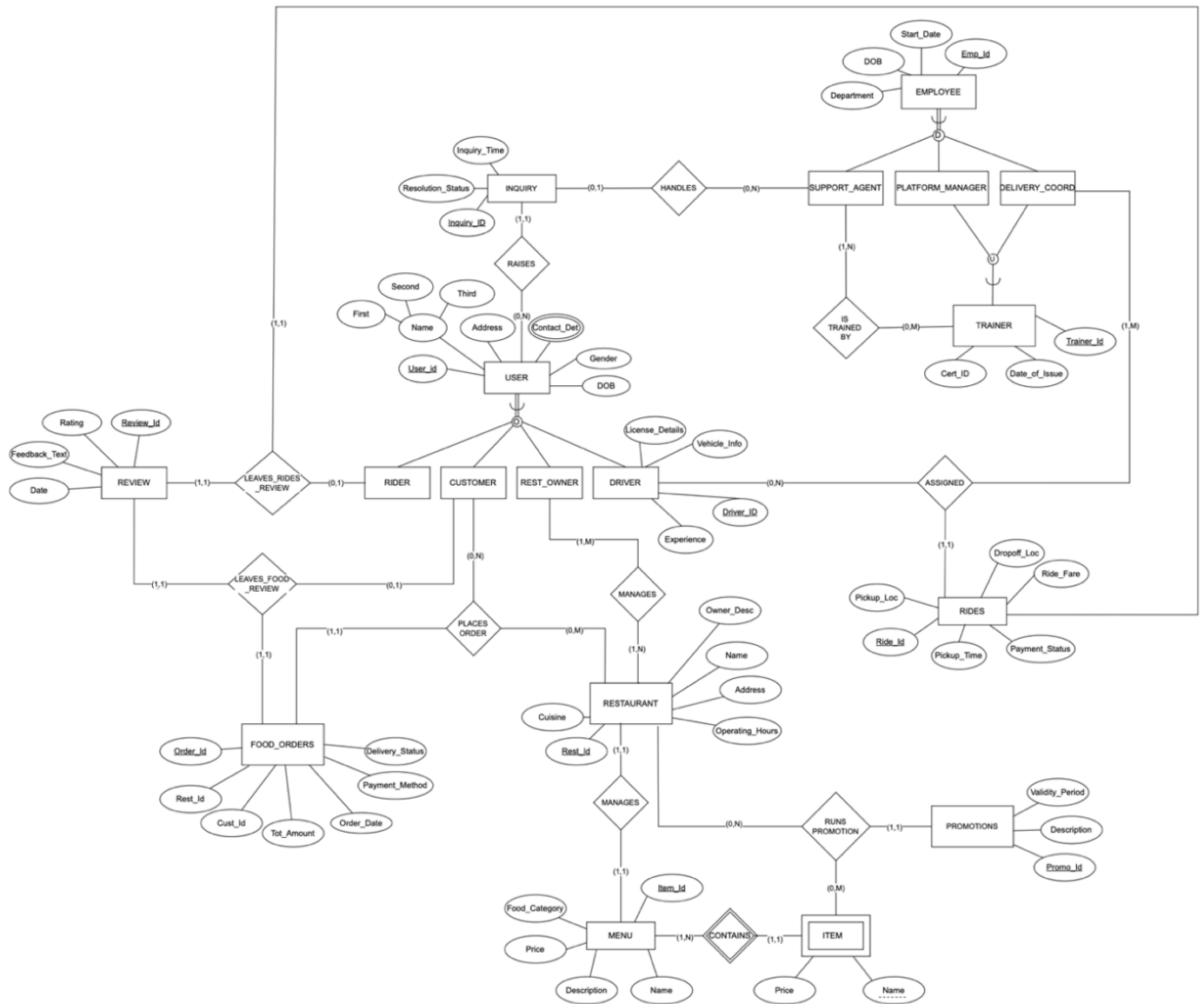
- Table partitioning for large transaction volumes
- Advanced indexing for performance
- High availability features for continuous operation

MySQL balances the need for scalability with maintaining data integrity - essential for a platform handling financial transactions and time-sensitive operations in a multi-stakeholder environment.

c. EER diagram with all assumptions.

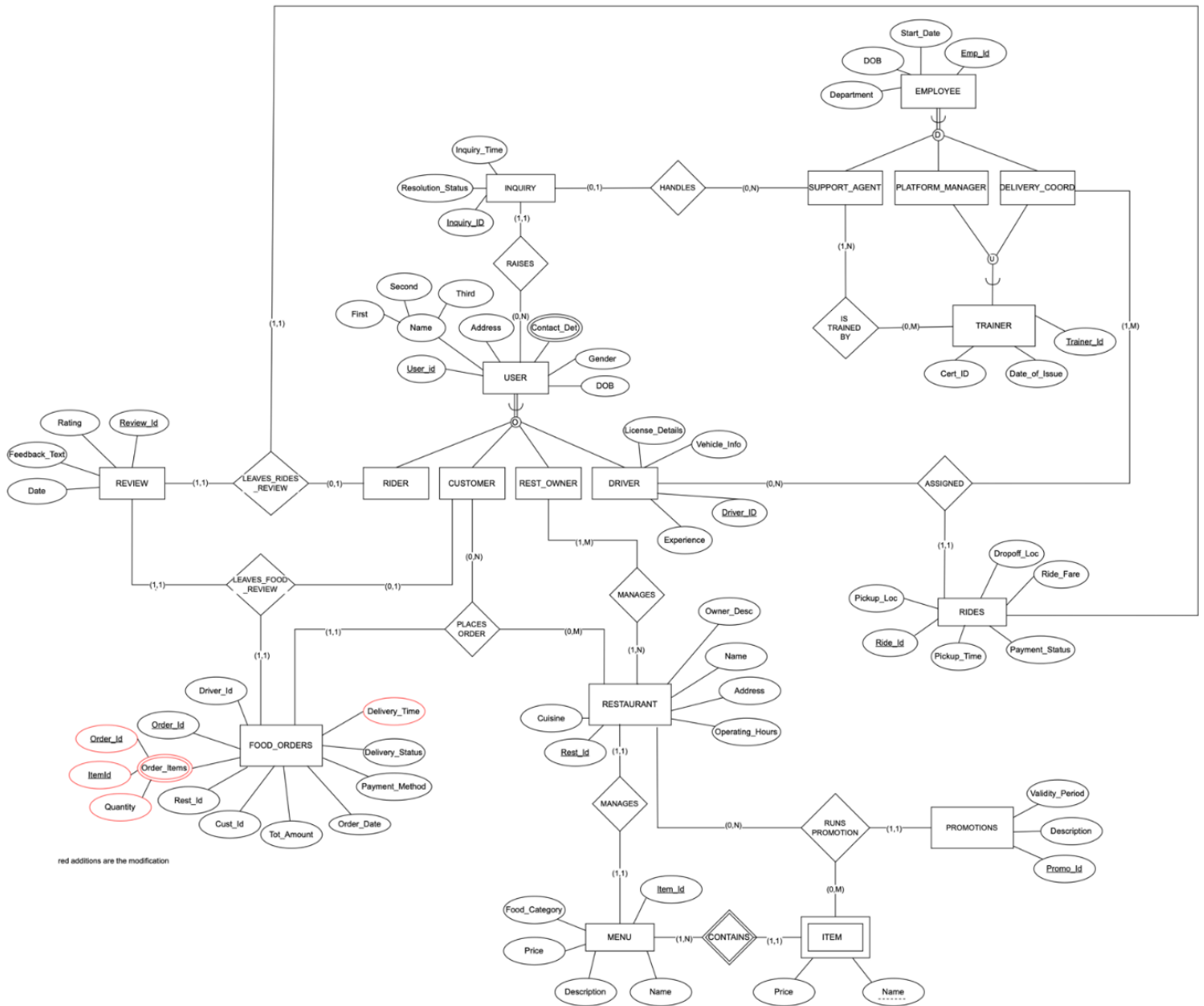
PHASE I

EER Model for GlobalRides System

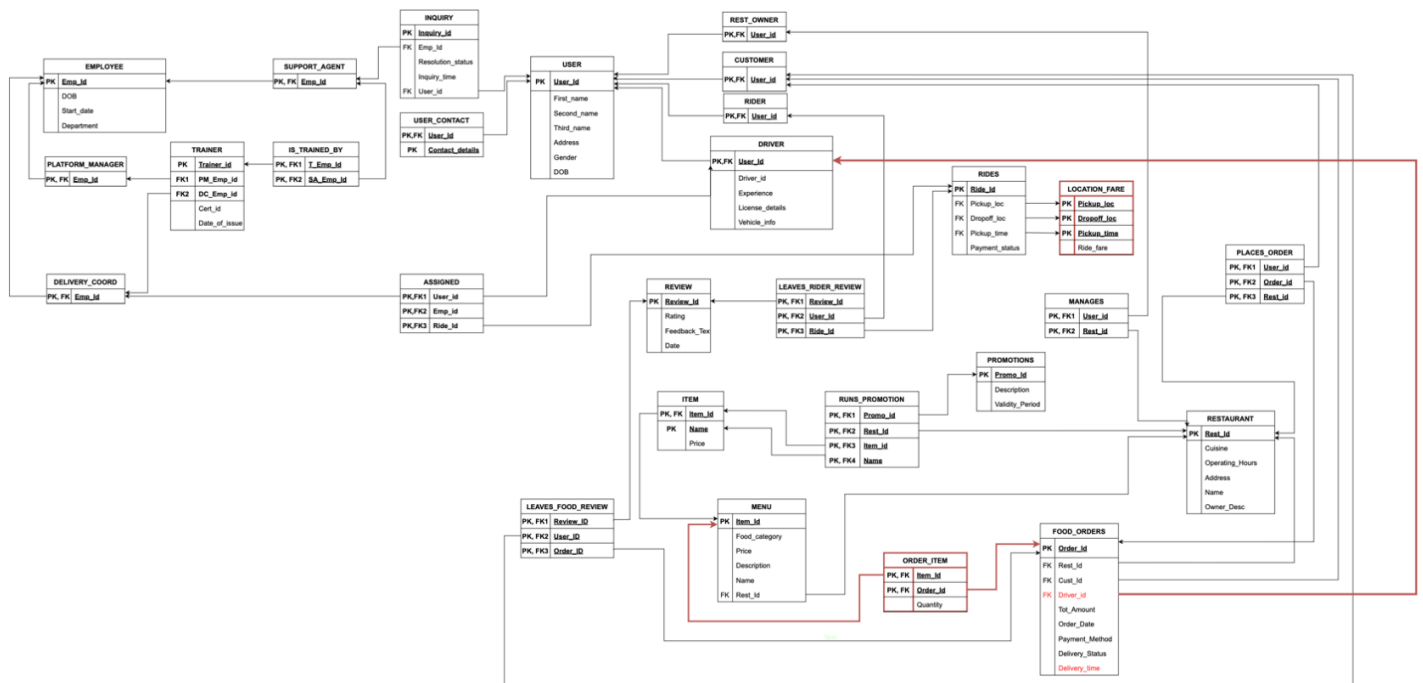


PHASE III

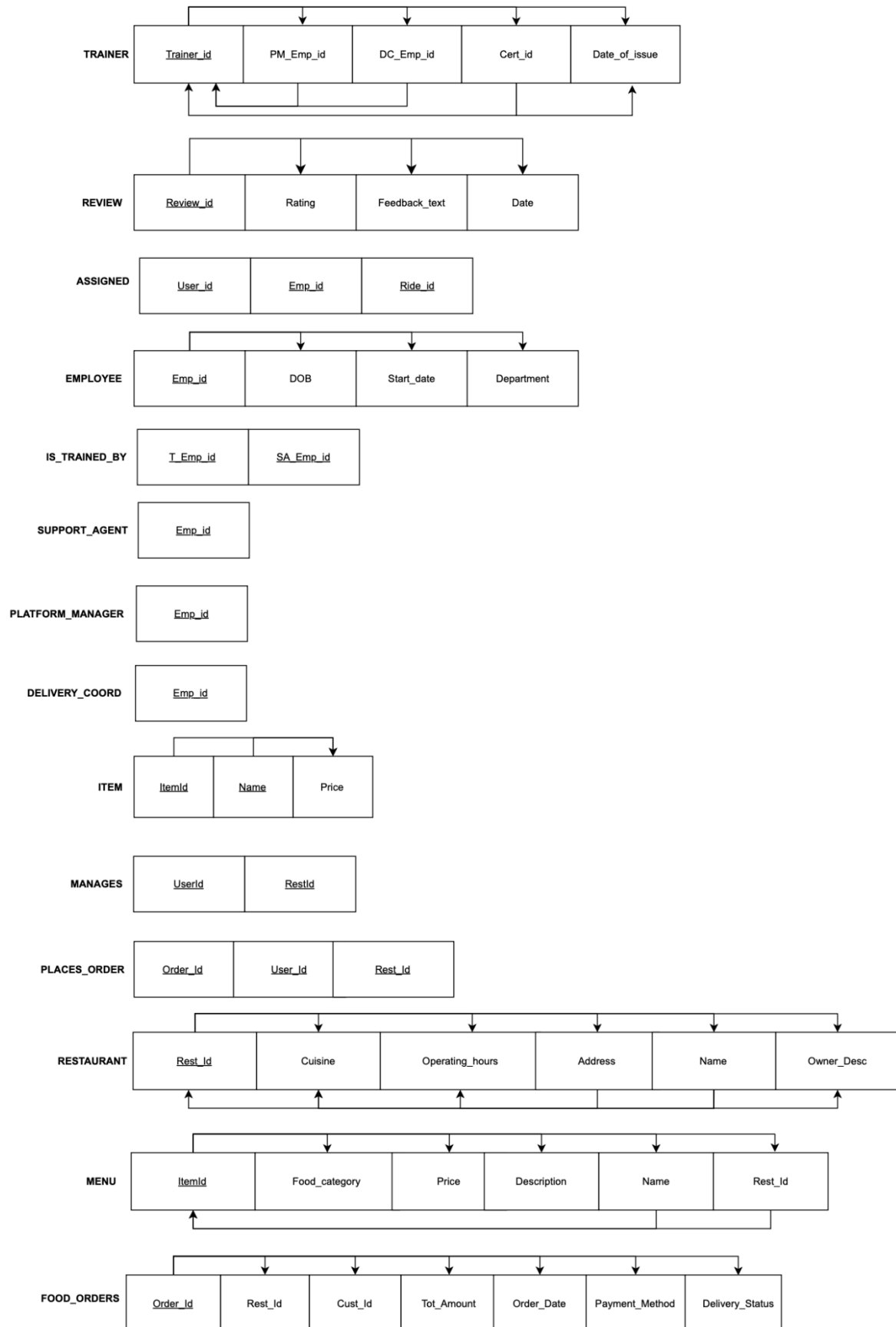
EER MODEL AFTER NORMALIZATION

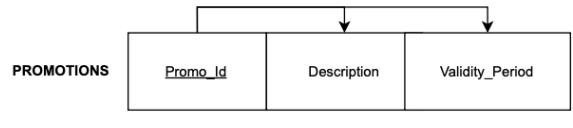
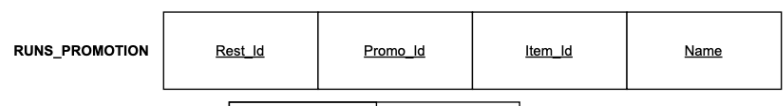
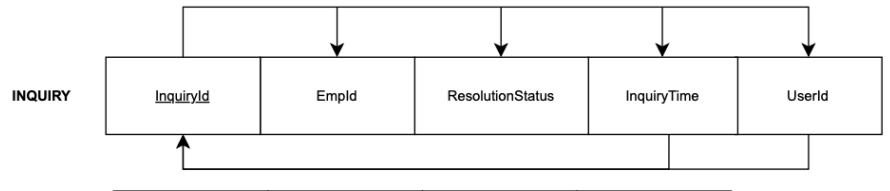
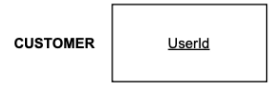
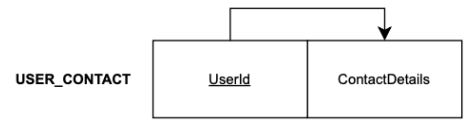
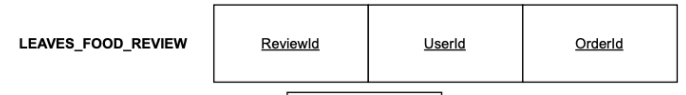
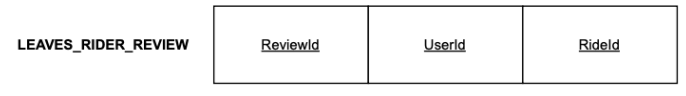
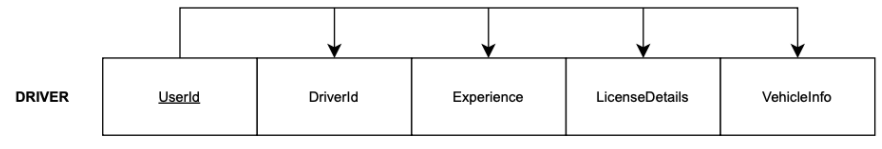
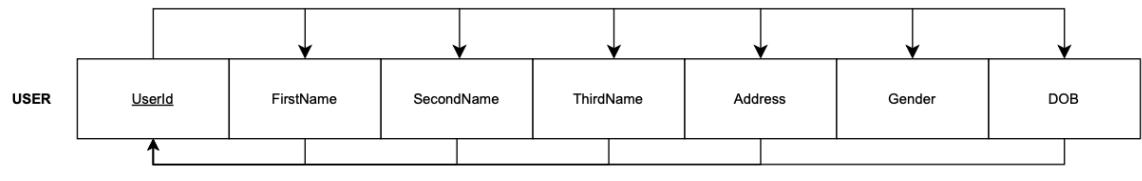
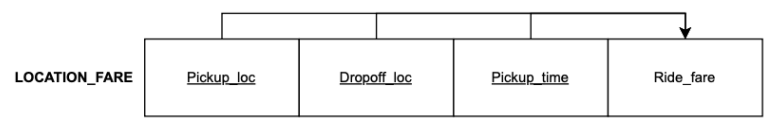
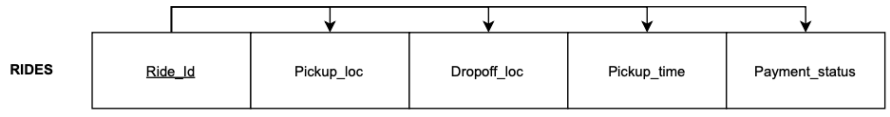


d. Relational Schema after normalization. Include primary and foreign keys for all relations.



e. Dependency diagrams.





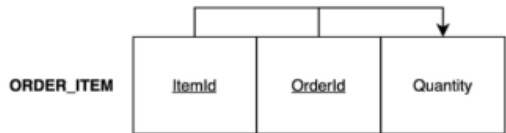
PHASE III

Dependency diagram changes after query design

Altered:



Added:



e. All SQL statements (from Phase III-c, d, and e)

- c) The tables for the relations and their attributes will be created and assigned as follows

use globalrides;

```
CREATE TABLE Employee
(empId INT PRIMARY KEY,
DOB DATE,
startDate DATE,
dept varchar(30));
```

```
CREATE TABLE User
(userId INT PRIMARY KEY,
fName varchar(20) NOT NULL,
sName varchar(20),
tName varchar(20),
address varchar(100),
gender varchar(20),
dob DATE);
```

```
CREATE TABLE SupportAgent
(empId INT PRIMARY KEY,
FOREIGN KEY(empId) REFERENCES Employee(empId));
```

```
CREATE TABLE Inquiry
(inquiryId INT PRIMARY KEY,
empId INT,
resolution_status varchar(30),
inquiryTime DATETIME,
userId INT,
FOREIGN KEY(userId) REFERENCES User(userId),
FOREIGN KEY(empId) REFERENCES SupportAgent(empId));
```

```
CREATE TABLE UserContact
(userId INT,
contactDetails varchar(30),
PRIMARY KEY(userId, contactDetails),
FOREIGN KEY(userId) REFERENCES User(userId))
```

```
CREATE TABLE Driver
(userId INT PRIMARY KEY,
driverId INT UNIQUE,
experience INT,
licenseDetails varchar(10) NOT NULL,
vehicleInfo varchar(10),
FOREIGN KEY(userId) REFERENCES User(userId));
```

```
CREATE TABLE Restaurant
(restId INT PRIMARY KEY,
cuisine varchar(15),
operatingHours varchar(20),
address varchar(100),
```

```
name varchar(20),  
ownerDesc varchar(50));
```

```
CREATE TABLE DeliveryCoordinator  
(empld INT PRIMARY KEY,  
FOREIGN KEY(empld) REFERENCES Employee(empld));
```

```
CREATE TABLE PlatformManager  
(empld INT PRIMARY KEY,  
FOREIGN KEY(empld) REFERENCES Employee(empld));
```

```
CREATE TABLE Trainer  
(trainerId INT PRIMARY KEY,  
pmEmpld INT,  
dcEmpld INT,  
certId INT UNIQUE NOT NULL,  
dateOfIssue DATE,  
FOREIGN KEY(pmEmpld) REFERENCES platformmanager(empld),  
FOREIGN KEY(dcEmpld) REFERENCES deliverycoordinator(empld));
```

```
CREATE TABLE IsTrainedBy  
(tEmpld INT,  
saEmpld INT,  
PRIMARY KEY(tEmpld, saEmpld),  
FOREIGN KEY(tEmpld) REFERENCES Trainer(trainerId),  
FOREIGN KEY(saEmpld) REFERENCES SupportAgent(empld));
```

```
CREATE TABLE Review  
(reviewId INT PRIMARY KEY,  
rating INT,  
feedbackText varchar(20),  
date DATE);
```

```
CREATE TABLE Rider  
(userId INT PRIMARY KEY,  
FOREIGN KEY(userId) REFERENCES User(userId));
```

```
CREATE TABLE Customer  
(userId INT PRIMARY KEY,  
FOREIGN KEY(userId) REFERENCES User(userId));
```

```
CREATE TABLE RestOwner  
(userId INT PRIMARY KEY,  
FOREIGN KEY(userId) REFERENCES User(userId));
```

```
CREATE TABLE Manages  
(userId INT,  
restId INT,  
PRIMARY KEY(userId, restId),  
FOREIGN KEY(userId) REFERENCES User(userId),  
FOREIGN KEY(restId) REFERENCES Restaurant(restId));
```

```
CREATE TABLE Promotions
(promold INT PRIMARY KEY,
description varchar(30),
validityPeriod varchar(20));
```

```
CREATE TABLE Menu
(itemId INT PRIMARY KEY,
foodCategory varchar(20),
price INT,
description varchar(30),
name varchar(20),
restId INT,
FOREIGN KEY(restId) REFERENCES Restaurant(restId));
```

```
CREATE TABLE FoodOrders
(orderId INT PRIMARY KEY,
restId INT,
custId INT,
totAmount INT,
orderDate DATE,
paymentMethod varchar(20),
deliveryStatus varchar(10),
FOREIGN KEY(restId) REFERENCES Restaurant(restId),
FOREIGN KEY(custId) REFERENCES Customer(userId));
```

```
CREATE TABLE Item
(itemId INT,
name varchar(30) NOT NULL,
price INT,
PRIMARY KEY(itemId, name),
FOREIGN KEY(itemId) REFERENCES Menu(itemId));
```

```
CREATE TABLE RunsPromotion
(restId INT,
promold INT,
itemId INT,
name varchar(30),
PRIMARY KEY(restId, promold, itemId),
FOREIGN KEY(promold) REFERENCES Promotions(promold),
FOREIGN KEY(itemId,name) REFERENCES Item(itemId,name),
FOREIGN KEY(restId) REFERENCES Restaurant(restId));
```

```
CREATE TABLE LeavesFoodReview
(reviewId INT,
userId INT,
orderId INT,
PRIMARY KEY(reviewId, userId, orderId),
FOREIGN KEY(reviewId) REFERENCES Review(reviewId),
FOREIGN KEY(userId) REFERENCES Customer(userId),
FOREIGN KEY(orderId) REFERENCES FoodOrders(orderId));
```

```
CREATE TABLE PlacesOrder
(orderId INT,
userId INT,
restId INT,
PRIMARY KEY(orderId,userId,restId),
FOREIGN KEY(orderId) REFERENCES FoodOrders(orderId),
FOREIGN KEY(userId) REFERENCES Customer(userId),
FOREIGN KEY(restId) REFERENCES Restaurant(restId));
```

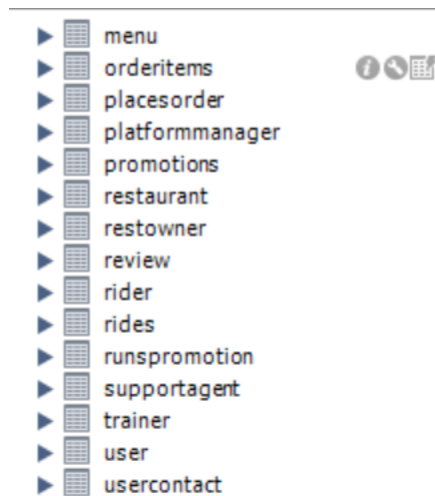
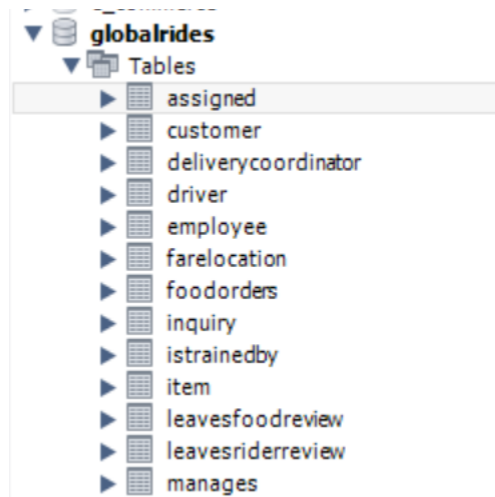
```
CREATE TABLE Rides
(rideId INT PRIMARY KEY,
pickUpLocation varchar(30) NOT NULL,
dropOffLocation varchar(30) NOT NULL,
pickUpTime TIME NOT NULL,
paymentStatus varchar(20),
FOREIGN KEY(pickUpLocation, dropOffLocation, pickUpTime) REFERENCES
FareLocation(pickUpLocation, dropOffLocation, pickUpTime));
```

```
CREATE TABLE FareLocation
(pickUpLocation varchar(30),
dropOffLocation varchar(30),
pickUpTime TIME,
rideFare INT,
PRIMARY KEY(pickUpLocation, dropOffLocation, pickUpTime));
```

```
CREATE TABLE Assigned
(userId INT,
emplId INT,
rideId INT,
PRIMARY KEY(userId, emplId, rideId),
FOREIGN KEY(userId) REFERENCES Driver(userId),
FOREIGN KEY(emplId) REFERENCES deliverycoordinator(emplId),
FOREIGN KEY(rideId) REFERENCES Rides(rideId));
```

```
CREATE TABLE LeavesRiderReview
(reviewId INT,
userId INT,
rideId INT,
PRIMARY KEY(reviewId, userId, rideId),
FOREIGN KEY(reviewId) REFERENCES Review(reviewId),
FOREIGN KEY(userId) REFERENCES Rider(userId),
FOREIGN KEY(rideId) REFERENCES Rides(rideId));
```

RESULTANT DATABASE WITH ALL THE CREATED TABLES.



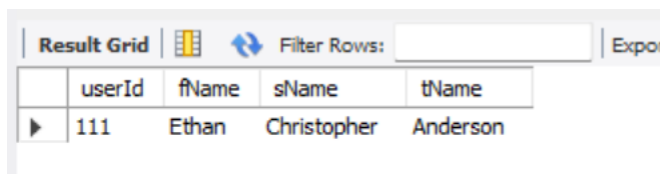
d) The design can be used to create the following views as described

1. LoyalCustomers: Which customers have placed orders consistently every month for the past year?

```
CREATE VIEW LoyalCustomers AS
SELECT      u.userId,
            u.fName,
            u.sName,
            u.tName
FROM User u
JOIN Customer c ON u.userId = c.userId
JOIN FoodOrders fo ON fo.custId = c.userId
WHERE fo.orderDate >= DATE_SUB(CURDATE(), INTERVAL 12 MONTH)
GROUP BY u.userId, u.fName, u.sName, u.tName
HAVING COUNT(DISTINCT DATE_FORMAT(fo.orderDate, '%Y-%m')) = 12;
```

Output

```
SELECT * FROM LoyalCustomers;
```



The screenshot shows a database interface with a 'Result Grid' tab. It contains a table with four columns: 'userId', 'fName', 'sName', and 'tName'. There is one data row with the values '111', 'Ethan', 'Christopher', and 'Anderson' respectively. Above the table, there are controls for 'Filter Rows' and an 'Export' button.

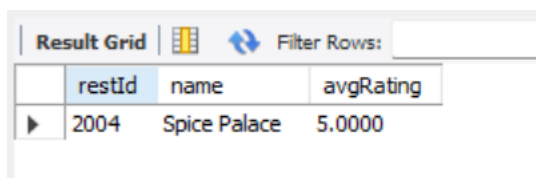
userId	fName	sName	tName
111	Ethan	Christopher	Anderson

2. TopRatedRestaurants: Which restaurants have an average review rating of 4.5 or higher over the past six months?

```
CREATE VIEW TopRatedRestaurants AS
SELECT r.restId, r.name, AVG(rv.rating) AS avgRating
FROM Review rv
JOIN LeavesFoodReview lfr ON rv.reviewId = lfr.reviewId
JOIN FoodOrders fo ON lfr.orderId = fo.orderId
JOIN Restaurant r ON fo.restId = r.restId
WHERE rv.date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY r.restId, r.name
HAVING AVG(rv.rating) >= 4.5;
```

Output

```
SELECT * FROM TopRatedRestaurants;
```



The screenshot shows a database interface with a 'Result Grid' tab. It contains a table with three columns: 'restId', 'name', and 'avgRating'. There is one data row with the values '2004', 'Spice Palace', and '5.0000' respectively. Above the table, there are controls for 'Filter Rows' and an 'Export' button.

restId	name	avgRating
2004	Spice Palace	5.0000

3. ActiveDrivers: Which delivery drivers have completed at least 20 deliveries within the last two weeks?

```
CREATE VIEW ActiveDrivers AS
SELECT d.userId, COUNT(*) AS deliveryCount
FROM FoodOrders fo
```

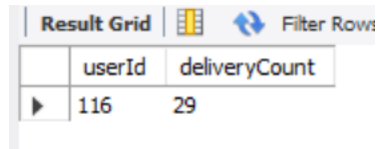
```

JOIN Driver d ON fo.driverId = d.userId
WHERE fo.orderDate >= DATE_SUB(CURDATE(), INTERVAL 14 DAY)
AND fo.driverId IS NOT NULL
GROUP BY d.userId
HAVING COUNT(*) >= 20;

```

Output

```
SELECT * FROM ActiveDrivers;
```



	userId	deliveryCount
▶	116	29

4. PopularMenuItems: What are the top 10 most frequently ordered menu items across all restaurants in the past three months?

[Modification :]

```

CREATE TABLE OrderItems (
  orderId INT,
  itemId INT,
  quantity INT DEFAULT 1,
  PRIMARY KEY(orderId, itemId),
  FOREIGN KEY(orderId) REFERENCES FoodOrders(orderId),
  FOREIGN KEY(itemId) REFERENCES Menu(itemId)
);

```

Output

Added table in globalrides database

```

CREATE VIEW PopularMenuItems AS
SELECT      m.itemId,
            m.name,
            COUNT(po.orderId) AS orderCount
FROM Menu m
JOIN FoodOrders fo ON fo.restId = m.restId
JOIN PlacesOrder po ON po.orderId = fo.orderId
WHERE fo.orderDate >= CURDATE() - INTERVAL 3 MONTH
GROUP BY m.itemId, m.name
ORDER BY orderCount DESC
LIMIT 10;

```

Output

```
SELECT * FROM PopularMenuItems;
```

Result Grid			
Filter Rows:			
	itemId	name	orderCount
▶	4014	Sashimi Platter	23
	4013	Edamame	23
	4015	Teriyaki Don	23
	4001	Bruschetta	20
	4002	Linguine Vongole	20
	4003	Tiramisu	20
	4004	Spring Rolls	20
	4005	Chilli Chicken	20
	4006	Mongolian Beef	20
	4022	Hot Wings	20

5. ProminentOwners: Which restaurant owners manage multiple restaurants with a combined total of at least 50 orders in the past month?

```
CREATE VIEW ProminentOwners AS
SELECT ro.userId AS ownerId,
       COUNT(DISTINCT m.restId) AS restaurantsManaged,
       COUNT(fo.orderId) AS totalOrders
FROM RestOwner ro
JOIN Manages m ON ro.userId = m.userId
JOIN Restaurant r ON m.restId = r.restId
JOIN FoodOrders fo ON m.restId = fo.restId
WHERE fo.orderDate >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
GROUP BY ro.userId
HAVING restaurantsManaged > 1 AND totalOrders >= 50;
```

Output

```
SELECT * FROM ProminentOwners;
```

Result Grid			
Filter Rows:			
	ownerId	restaurantsManaged	totalOrders
▶	115	3	55

e) The query questions and their respective SQL queries are given as follows:



1. TopEarningDrivers: List the names and total earnings of the top five drivers.

```
SELECT      u.userId,
            u.fName,
            u.sName,
            u.tName,
            SUM(fl.rideFare) AS totalEarnings
FROM User u
JOIN Driver d ON u.userId = d.userId
JOIN Assigned a ON a.userId = d.userId
JOIN Rides r ON r.rideId = a.rideId
JOIN FareLocation fl ON fl.pickUpLocation = r.pickUpLocation AND
                    fl.pickUpTime = r.pickUpTime AND
                    fl.dropOffLocation = r.dropOffLocation
```

```
GROUP BY u.userId, u.fName, u.sName, u.tName
ORDER BY totalEarnings DESC
LIMIT 5;
```

Output

Result Grid

Filter Rows:

Export:



	userId	fName	sName	tName	totalEarnings
▶	101	John	Michael	Smith	132
	102	Emma	Rose	NULL	90
	117	Alexander	James	Miller	53
	116	Mia	NULL	Wilson	48
	115	Matthew	Andrew	NULL	47

2. HighSpendingCustomers: Identify customers who have spent more than \$1,000 and list their total expenditure.



```
SELECT      u.userId,
            u.fName,
            u.sName,
            u.tName,
            SUM(fo.totAmount) AS totalExpenditure
FROM User u
JOIN Customer c ON u.userId = c.userId
JOIN PlacesOrder po ON po.userId = c.userId
JOIN FoodOrders fo ON fo.orderId = po.orderId
GROUP BY u.userId, u.fName, u.sName, u.tName
HAVING totalExpenditure > 1000;
```

Output

Result Grid



Filter Rows:

Export:

	userId	fName	sName	tName	totalExpenditure
▶	108	Ava	Elizabeth	Davis	1054
	109	Michael	Joseph	NULL	1070
	111	Ethan	Christopher	Anderson	2260

3. FrequentReviewers: Find customers who have left at least 10 reviews and their average review rating.

```
SELECT      u.userId,
            u.fName,
            u.sName,
            u.tName,
            COUNT(lfr.reviewId) AS reviewCount,
            AVG(r.rating) AS averageRating
FROM User u
JOIN Customer c ON c.userId = u.userId
JOIN LeavesFoodReview lfr ON lfr.userId = c.userId
JOIN Review r ON lfr.reviewId = r.reviewId
GROUP BY u.userId, u.fName, u.sName, u.tName
HAVING reviewCount >= 10;
```

Output

Result Grid

Filter Rows:

Export:

Wrap Cell Conte

	userId	fName	sName	tName	reviewCount	averageRating
▶	111	Ethan	Christopher	Anderson	11	3.7273

4. InactiveRestaurants: List restaurants that have not received any orders in the past months.

```
SELECT r.restId,  
       r.name,  
       MAX(fo.orderDate) as lastOrderDate  
FROM Restaurant r  
JOIN FoodOrders fo ON fo.restId = r.restId  
GROUP BY r.restId, r.name  
HAVING MAX(fo.orderDate) < (CURDATE() - INTERVAL 2 MONTH) OR MAX(fo.orderDate) IS NULL;
```

Output

	Result Grid	Filter Rows:	E
	restId	name	lastOrderDate
▶	2011	Saigon Corner	NULL
	2012	Athens Taverna	2025-01-05 18:30:00

5. PeakOrderDay: Identify the day of the week with the highest number of orders in the past month.

```
SELECT DAYNAME(fo.orderDate) AS dayOfWeek,  
       COUNT(*) AS orderCount  
FROM FoodOrders fo  
WHERE (fo.orderDate) >= (CURDATE() - INTERVAL 1 MONTH)  
GROUP BY DAYNAME(fo.orderDate)  
ORDER BY orderCount DESC  
LIMIT 1;
```

Output

	Result Grid	Filter Rows:
	dayOfWeek	orderCount
▶	Saturday	24

6. HighEarningRestaurants: Find the top three restaurants with the highest total revenue in the past year.

```
SELECT r.restId,  
       r.name,  
       SUM(fo.totAmount) AS totalRevenue  
FROM Restaurant r  
JOIN FoodOrders fo ON fo.restId = r.restId  
WHERE (fo.orderDate) >= (CURDATE() - INTERVAL 1 YEAR)  
GROUP BY r.restId, r.name  
ORDER BY totalRevenue DESC  
LIMIT 3;
```

Output

Result Grid

Filter Rows:

	restId	name	totalRevenue
▶	2005	Sakura Zen	1421
	2001	Bella Vista	1149
	2007	Blue Aegean	1132

7. PopularCuisineType: Identify the most frequently ordered cuisine type in the past six months.

```
SELECT r.cuisine,  
       COUNT(*) AS cuisineCount  
FROM Restaurant r  
JOIN FoodOrders fo ON fo.restId = r.restId  
WHERE (fo.orderDate) >= (CURDATE() - INTERVAL 6 MONTH)  
GROUP BY r.cuisine  
ORDER BY cuisineCount DESC  
LIMIT 1;
```

Output

Result Grid	Filter Rows:
cuisine	cuisineCount
Japanese	23

8. LongestRideRoutes: Identify the top five ride routes with the longest travel distances.

[Assumption: Assume higher rideFare would cover longer distances]

```
SELECT pickUpLocation, dropOffLocation, rideFare  
FROM FareLocation  
ORDER BY rideFare DESC  
LIMIT 5;
```

Output

Result Grid	Filter Rows:	Export:
pickUpLocation	dropOffLocation	rideFare
Times Square	JFK Airport	70
JFK Airport	Times Square	65
Brooklyn Bridge	Madison Square Garden	35
Madison Square Garden	Brooklyn Bridge	35
Wall Street	Central Park	32

9. DriverRideCounts: Display the total number of rides delivered by each driver in the past three months.

```
SELECT u.userId,  
       u.fName,  
       u.sName,  
       u.tName,  
       COUNT(r.rideId) AS totRides  
FROM User u  
JOIN Driver d ON d.userId = u.userId
```

```

JOIN Assigned a ON a.userId = d.userId
JOIN Rides r ON r.ridId = a.ridId
WHERE r.pickUpTime >= (CURDATE() - INTERVAL 3 MONTH)
GROUP BY u.userId, u.fName, u.sName, u.tName;

```

Output

	userId	fName	sName	tName	totRides
▶	101	John	Michael	Smith	3
	102	Emma	Rose	NULL	2
	115	Matthew	Andrew	NULL	2
	116	Mia	NULL	Wilson	2
	117	Alexander	James	Miller	2

10. UndeliveredOrders: Find all orders that were not delivered within the promised time window and their delay durations.

[Assumption: the promised delivery time is between 30 - 40 mins after the orderDate]

```

SELECT orderId,
       orderDate,
       deliveryTime,
       TIMESTAMPDIFF(MINUTE, orderDate, deliveryTime) AS delayMinutes
FROM FoodOrders
WHERE deliveryTime IS NOT NULL
      AND TIMESTAMPDIFF(MINUTE, orderDate, deliveryTime) > 40
      AND deliveryStatus = "Delivered";

```

Output

	orderId	orderDate	deliveryTime	delayMinutes
▶	7001	2024-05-15 18:30:00	2024-05-15 19:15:00	5
	7005	2024-09-12 19:45:00	2024-09-12 20:30:00	5
	7008	2024-12-08 19:30:00	2024-12-08 20:20:00	10
	7009	2025-04-26 20:30:00	2025-04-26 21:15:00	5
	7013	2025-04-30 20:15:00	2025-04-30 21:00:00	5
	7014	2025-05-01 18:30:00	2025-05-01 19:15:00	5
	7023	2025-05-09 20:30:00	2025-05-09 21:15:00	5
	7033	2024-07-20 20:00:00	2024-07-20 20:45:00	5
	7034	2024-08-20 19:30:00	2024-08-20 20:15:00	5
	7038	2024-12-20 19:00:00	2024-12-20 19:50:00	10
	7039	2025-01-20 20:00:00	2025-01-20 20:45:00	5
	7041	2025-03-20 19:30:00	2025-03-20 20:15:00	5
	7051	2025-05-05 19:30:00	2025-05-05 20:15:00	5
	7057	2025-05-10 09:00:00	2025-05-10 09:45:00	5

7102	2025-01-05 18:30:00	2025-01-05 19:15:00	5
7205	2025-03-28 19:00:00	2025-03-28 19:45:00	5
7210	2025-04-09 19:30:00	2025-04-09 20:15:00	5
7234	2025-03-22 19:30:00	2025-03-22 20:15:00	5
7236	2025-03-28 19:15:00	2025-03-28 20:00:00	5
7240	2025-04-09 20:00:00	2025-04-09 20:45:00	5
7250	2025-05-03 20:00:00	2025-05-03 20:45:00	5

Result 15 ×

11. MostCommonPaymentMethods: Identify the most frequently used payment method on the platform for both rides and food orders.

[Assumption: paymentStatus also contains the payment information for rides]

```
SELECT method, SUM(count) AS totalUsage
FROM (
  SELECT paymentMethod AS method,
    COUNT(*) AS count
  FROM FoodOrders
  GROUP BY paymentMethod

  UNION ALL

  SELECT paymentStatus AS method,
    COUNT(*) AS count
  FROM Rides
  GROUP BY paymentStatus
) AS combinedMethods
GROUP BY method
ORDER BY totalUsage DESC;
```

Output

Result Grid	Filter Rows:
method	totalUsage
▶ Credit Card	41
Cash	23
PayPal	22
Debit Card	20
Apple Pay	18
Google Pay	18
COMPLETED	7
PENDING	2
FAILED	1
PROCESSING	1

12. TrainingCount: List the number of Support Agents trained by each Trainer and their respective certification dates.

```
SELECT t.trainerId,
  t.certId,
  t.dateOfIssue,
  COUNT(sa.emplId) AS totSaTrained
FROM Trainer t
JOIN IsTrainedBy itb ON itb.tEmplId = t.trainerId
```



```
JOIN SupportAgent sa ON sa.emplId = itb.saEmplId
GROUP BY t.trainerId, t.certId, t.dateOfIssue;
```

Output

	trainerId	certId	dateOfIssue	totSaTrained
▶	3001	70001	2023-08-15	2
	3002	70002	2023-12-20	2
	3003	70003	2024-03-25	2
	3004	70004	2024-09-10	1

13. MultiRoleUsers: Identify users who simultaneously serve as both Drivers and Restaurant Owners, along with their details.

```
SELECT u.userId,
       u.fName,
       u.sName,
       u.tName
FROM User u
JOIN Driver d ON u.userId = d.userId
JOIN RestOwner ro ON ro.userId = d.userId;
```

Output

	userId	fName	sName	tName
▶	115	Matthew	Andrew	NULL
	116	Mia	NULL	Wilson
	117	Alexander	James	Miller

14. DriverVehicleTypes: Display the distribution of drivers by vehicle type (Sedan, SUV, and etc.), including the total number for each type.

```
SELECT vehicleInfo,
       COUNT(userId) AS totalDrivers
FROM Driver
GROUP BY vehicleInfo;
```


Output

	vehicleInfo	totalDrivers
▶	Honda CBR	1
	Tesla S	2
	Yamaha MT	1
	Honda Civ	1
	Suzuki GS	1

15. HighlyRatedDrivers: Find drivers who have an average customer review rating of at least 4.8 and the number of rides they've completed

```
SELECT d.driverId,  
       COUNT(r.rideId) AS numRides,  
       AVG(rv.rating) AS avgRating  
FROM Driver d  
JOIN LeavesRiderReview lrr ON d.userId = lrr.userId  
JOIN Rides r ON lrr.rideId = r.rideId  
JOIN Review rv ON lrr.reviewId = rv.reviewId  
GROUP BY d.driverId  
HAVING AVG(rv.rating) >= 4.8;
```

Output

Result Grid			Filter Rows:
	driverId	numRides	avgRating
▶	8001	2	5.0000
	8006	1	5.0000

16. TopPerformingPromotions: Identify the top five promotions that resulted in the highest percentage increase in order volume during this month compared to the last month without the promotion, and display the percentage increase along with the promotion details.

```
WITH PromoOrders AS (  
    SELECT rp.promoId,  
           fo.orderId,  
           fo.orderDate  
    FROM RunsPromotion rp  
    JOIN OrderItems oi ON rp.itemId = oi.itemId  
    JOIN FoodOrders fo ON oi.orderId = fo.orderId),  
MonthlyCounts AS (  
    SELECT promoId,  
           CASE  
               WHEN MONTH(orderDate) = MONTH(CURDATE()) THEN 'thisMonth'  
               WHEN MONTH(orderDate) = MONTH(DATE_SUB(CURDATE(), INTERVAL 1 MONTH)) THEN 'lastMonth'  
               ELSE NULL  
           END AS monthCategory,  
           COUNT(DISTINCT orderId) AS orderCount  
    FROM PromoOrders  
    WHERE orderDate >= DATE_SUB(CURDATE(), INTERVAL 2 MONTH)  
    GROUP BY promoId, monthCategory  
)  
,  
PivotCounts AS (  
    SELECT promoId,  
           MAX(CASE WHEN monthCategory = 'thisMonth' THEN orderCount ELSE 0 END) AS thisMonthCount,  
           MAX(CASE WHEN monthCategory = 'lastMonth' THEN orderCount ELSE 0 END) AS lastMonthCount  
    FROM MonthlyCounts  
    GROUP BY promoId  
)  
,  
WithPercentageIncrease AS (  
    SELECT  
        pc.promoId,  
        p.description,
```

```

p.validityPeriod,
thisMonthCount,
lastMonthCount,
CASE
    WHEN lastMonthCount = 0 THEN 100.0
    ELSE ROUND(((thisMonthCount - lastMonthCount) / lastMonthCount) * 100, 2)
END AS percentageIncrease
FROM PivotCounts pc
JOIN Promotions p ON pc.promold = p.promold
)

SELECT *
FROM WithPercentageIncrease
ORDER BY percentageIncrease DESC
LIMIT 5;

```

Output

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	promoId	description	validityPeriod	thisMonthCount	lastMonthCount	percentageIncrease
	9002	Free Delivery Weekend	2025-05-10 to 05-12	11	3	266.67
	9008	Free Drink with Meal	2025-05-25 to 06-10	13	4	225.00
	9007	Summer Special 25% Off	2025-06-20 to 08-31	7	4	75.00
	9003	\$5 Off Orders Over \$30	2025-05-15 to 06-15	3	2	50.00
	9001	20% Off First Order	2025-05-01 to 05-31	11	8	37.50

17. OrderTimeAnalysis: Identify the average order delivery time for each restaurant in the past month and list those with an average time exceeding 30 minutes.

[Assumptions: In FoodOrders table, the orderDate includes the time that the order was placed as well.

Modification: In FoodOrders, we added an attribute deliveryTime]

```

SELECT fo.restId,
       r.name AS restaurantName,
       ROUND(AVG(TIMESTAMPDIFF(MINUTE, fo.orderDate, fo.deliveryTime)), 2) AS avgDeliveryTimeMins
FROM FoodOrders fo
JOIN Restaurant r ON fo.restId = r.restId
WHERE fo.deliveryTime IS NOT NULL
      AND fo.orderDate >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH)
GROUP BY fo.restId, r.name
HAVING avgDeliveryTimeMins > 30;

```

Output

Result Grid			 Filter Rows:	
	restId	restaurantName	avgDeliveryTimeMins	
▶	2001	Bella Vista	38.42	
	2002	Golden Dragon	40.28	
	2004	Spice Palace	40.00	
	2005	Sakura Zen	39.72	
	2007	Blue Aegean	35.00	
	2008	Stars & Stripes	30.56	
	2010	Seoul Kitchen	40.00	